Combining Geometry and Domain Knowledge to Interpret Hand-Drawn Diagrams

Leslie M. Gennari
Mechanical Engineering Dept.
Carnegie Mellon University
Pittsburgh, PA 15213

Levent Burak Kara
Mechanical Engineering Dept.
Carnegie Mellon University
Pittsburgh, PA 15213
lkara@andrew.cmu.edu

Thomas F. Stahovich
Mechanical Engineering Dept.
University of California
Riverside, CA 92521
stahov@engr.ucr.edu

Abstract

We present a sketch understanding system for network-like diagrams consisting of symbols linked together. This system employs a novel parser to automatically extract symbols from a continuous stream of pen strokes. The parser uses geometric information to enumerate candidate symbols, and then uses domain knowledge to prune away unlikely candidates. The candidates are classified with a novel, domain-independent, probabilistic, feature-based symbol recognizer. Domain knowledge and context are used to correct parsing and recognition errors. To demonstrate our system, we used it to create a sketch-based interface for an electric circuit analysis program.

Introduction

Sketching with pencil and paper has long been an important means of communication and problem-solving for designers and engineers. There are a variety of reasons for this. For example, sketches are a convenient tool for examining geometric, temporal, and other similar relationships which cannot easily be described in words. Likewise, the simplicity and ease of creating a sketch allows one to focus on problem solving rather than the communication medium. Yet, despite the importance of sketches in engineering practice, traditional engineering software can do little with them. Engineers often find themselves recreating their sketches on the computer in order to take advantage of such software. We are working to change this by creating sketch understanding techniques that enable software to work directly from the kinds of sketches engineers ordinarily draw.

This work addresses three key problems associated with achieving natural, sketch-based user interfaces. The first has to do with *ink parsing*, the task of automatically grouping a user's pen strokes into clusters representing the intended symbols. Many current sketch interpretation systems avoid this problem by requiring the user to explicitly indicate the intended partitioning of the ink. This is often done by pressing a button on the stylus or by pausing between symbols (Apte *et al.*, 1993; Narayanaswamy, 1996; Fonseca *et al.*, 2002). Other systems require each object to be drawn in a single pen stroke (Rubine, 1991; Landay and Myers, 2001). Unfortunately, such constraints on the way the user draws often result in a less than natural drawing environment.

Prior to parsing, the pen strokes are segmented into lines and arcs. A combination of geometric and domain-specific knowledge is then used to locate symbols. Our parsing approach allows for multiple symbols to be drawn in the same stroke, and allows individual symbols to be drawn in multiple strokes.

The second problem we address concerns the task of *symbol recognition*. Many current recognizers are limited by their sensitivity to size, orientation, or the number of strokes used to draw a symbol (Rubine, 1991; Lee, 1992). Additionally, many recognizers require a significant amount of training data or must be hard-coded (Apte *et al.*, 1993; Alvarado, 2000). Our recognizer uses a probabilistic, feature-based approach designed to handle the kinds of variations common in hand-drawn sketches. Additionally, it is insensitive to rotations and scaling, the parts of a symbol can be drawn in any order, symbols can be drawn with multiple pen strokes, and a single pen stroke can contain multiple symbols. Finally, our recognizer requires only a few training examples to reliably classify symbols.

The third problem we address is the use of context to automatically correct errors. Due to the ambiguity inherent in sketches, it is difficult to achieve perfect parsing and recognition accuracy. Our sketch interpreter employs automatic error correction techniques to help fix typical errors. Once a sketch has been interpreted, domain knowledge is used to determine if the interpretation of the sketch is self-consistent. If not, parsing and recognition are revisited so as to eliminate the inconsistencies.

Our system is designed to work for network-like diagrams containing isolated, non-overlapping symbols that are linked together. Examples include analog electric circuits, data flow diagrams, and algorithmic flowcharts. As an illustration of our system's capabilities, we developed a sketch-based interface for the SPICE electric circuit analysis program. The electrical circuit domain was chosen because it provides an adequate level of complexity to demonstrate our work. Our interface is called AC-SPARC for **A**nalog **C**ircuit **S**ketch **PA**rsing, **R**ecognition, and error **C**orrection.

System Overview

Our system is designed for use with a digitizing tablet and stylus, or other similar hardware. Here, we use the Wacom Cintiq LCD tablet because it enables the user to draw directly on the computer display. As the user draws a circuit, the ink is segmented into line and arc segments. The user can choose to view the sketch in its raw form or in the cleaned-up segmented form. Once the sketch is complete, the user selects a menu with the stylus causing the program to interpret the sketch. The identified electrical components are then indicated with color coding and text labels as shown in Figure 1. An input file for SPICE is also generated.

Our interface provides an easy means for correcting common interpretation errors. If the program fails to locate a symbol, the user can explicitly mark it by holding down a stylus button and circling the symbol. If any non-symbol ink is mistakenly identified as a symbol, the user need simply draw a diagonal line through it while pressing a stylus button. If a symbol is misclassified, the user can tap the stylus on it while holding the stylus button. A dialog box will appear containing a list of alternative classifications which can be selected with the stylus.

Our interface allows symbols to be easily added to or removed from the sketch as the design evolves. Users can erase ink with the eraser end of the stylus, just as one would with a pencil eraser. Wires and symbols can be added to the sketch at any time by simply drawing them.

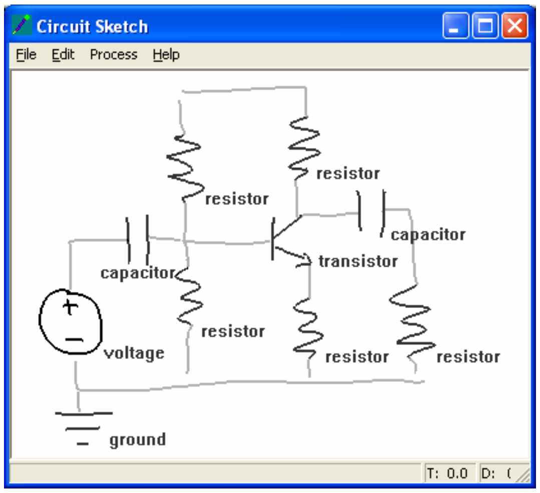

Figure 1: AC-SPARC interface. Recognized circuit components are indicated with color coding and text labels.

Technical Details

Our approach to understanding a sketch is based on the architecture shown in Figure 2. The first step involves decomposing the users' pen strokes into line and arc segments that closely match the original ink. This process, called *ink segmentation*, provides compact descriptions of the pen strokes that facilitate parsing and recognition. Next, geometric tests are used to locate candidate symbols, which are then classified using our symbol recognizer. Knowledge about the particular domain of the sketch is then used to prune the list of candidate symbols. Finally, domain knowledge is used to automatically correct errors made in the previous steps. This process results in a final interpretation of the sketch, which the user can edit if necessary.

The following sections describe each of these steps in detail. Note, however, that for the sake of continuity in the discussion, we present both parsing steps before presenting the symbol recognizer.

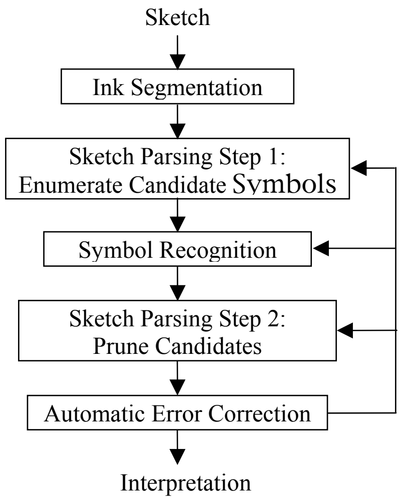

Figure 2: Architecture of our sketch interpreter.

Segmentation

As the user draws, the digital ink is segmented into line and arc segments so as to facilitate parsing and recognition. The segmentation process involves searching along the pen stroke for "segment points," points that divide the stroke into different geometric primitives (Stahovich, 2004). These points are distinguished by both the motion of the pen tip observed while the strokes were drawn, and the shape of the resulting ink. Segment points are generally points at which the pen speed is at a minimum, the ink exhibits high curvature, or the sign of the curvature changes. Once the segment points have been identified, a least squares approach is used to fit lines and arcs to the ink.

Sketch Parsing

The goal of parsing is to identify the sets of line and arc segments that comprise individual symbols. In the electrical circuit domain, the symbols are circuit components. Parsing is concerned only with locating the symbols; classifying each symbol is the task of our symbol recognizer. Our parsing approach begins by using geometric information to identify candidate symbols. Domain knowledge, along with our symbol recognizer, is then used to identify which of the candidates actually are symbols. This parser is intended for diagrams consisting of symbols connected by wires, arrows, or other similar connectors.

Parsing Step 1: Enumerating Candidate Symbols: We assume that the user finishes drawing one symbol before drawing a wire or another component. Our observations of people drawing suggest that this assumption is reasonable, especially for electrical circuits. Therefore, when locating candidate symbols, we need to consider only consecutively drawn segments. To further reduce the search space, we also establish limits on the number of segments that a symbol may contain. The lower limit is two, since it is uncommon that a symbol is represented by a single line or arc segment. The upper limit depends on the particular user's drawing style, and is determined by examining the user's training data for the symbol recognizer. In practice, this number is typically between 6 and 12. Candidate symbols are, therefore, groups of time-ordered segments containing between two and some user-dependent maximum number of segments.

Candidate symbols are enumerated using two types of geometric tests to identify possible starts and ends of symbols. The first test looks for regions in which there is a high concentration of ink. The second looks for changes in the characteristics of the segments, such as when a long segment is followed by a much shorter segment. These tests are described in detail below.

Ink Density Locator: Symbols usually consist of a high concentration of ink, while the ink of connectors is often more spread out. Our ink density approach identifies candidate symbols by searching for regions of high ink density. More specifically, *we search for sequences of segments having the property that the addition of another segment to either end of the sequence causes a decrease in density*, as this is an indication of adding a connector segment. We define *ink density* as the ratio of the square of the ink length to the area of the oriented bounding box of the ink:

$$density = \frac{ink\_length^2}{bounding\_box\_area}$$

Here, in addition to the actual ink shown on the screen, the ink length also includes the *hidden ink*, which we define as the ink that would occur if the user did not pick up the stylus while drawing. For example, the hidden ink of a voltage source is shown by the dotted lines in Figure 3. Including the hidden ink accentuates the density of symbols drawn with multiple strokes, thus making them easier to identify. We square the ink length so that it scales the same way as bounding box area, thus making the density parameter insensitive to uniform scaling. The bounding box is the smallest rectangle, not necessarily aligned with the coordinate axes, containing all of the segments in question.

A symbol consists of a sequence of line and arc segments, beginning with a start segment and ending with an end segment. The ink density analysis uses a forward-backward algorithm to find the start and end segments. The forward step is used to find the end segment of a symbol by finding segments whose addition to the sequence significantly decrease the sequence's density. In the backward step, the best start segments are located for each end segment, again by looking for decreases in density. (The approach is also repeated with time reversed. In the forward pass, the starts of symbols are identified, and in the backward pass, the corresponding ends are identified.)

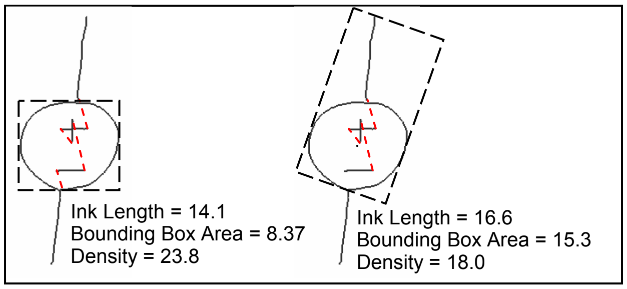

Figure 3: Density decreases when segment is added to symbol. Hidden ink is shown dotted. Bounding box is dashed.

Consider applying this approach to the resistor shown in Figure 4. For sake of example, assume that in the forward step we are starting from Segment 5. The initial sequence consists of Segments 5 and 6. Adding Segments 7, 8, 9, and 10 results in density changes of 87.4%, -5.1%, -35.5%, and 10.1% respectively. Only the addition of Segment 9 produces a significant density decrease (-35.5%), indicating that only Segment 8 is a possible end segment. If multiple segments had resulted in significant decreases, then multiple ends would have been identified. In the backward step, the sequence initially consists of Segments 8 and 7. Segments are then added to the start of this sequence until Segment 2 is reached. This results in density changes of 42.9%, 20.7%, 13.0%, -2.1%, and -23.0%. The addition of Segment 2 causes the biggest decrease in density, and thus Segment 3 is considered the best start segment for the sequence that ends with Segment 8. The result is a candidate symbol consisting of Segments 3-8, which in fact corresponds to the intended resistor. It is interesting to note that the sequence of segments from 4-8 actually had a higher density than the sequence from 3-8. However, the density of the former would not have decreased significantly with the addition of another segment to its start, and so it was not considered a candidate symbol.

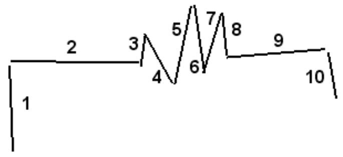

Figure 4: Sketch of a resistor used to illustrate the density method for locating symbols.

Segment Difference Locator: There are usually large differences between a connector and the first segment of a symbol, and between the last segment of a symbol and the subsequent connector. Our second symbol locator finds symbols by identifying those differences.

For each segment in the sketch, we calculate four characteristics. These include segment type (line or arc), segment length, segment orientation (angle relative to the previous segment), and type of intersection between the current segment and the previous segment (endpoint-to-endpoint, endpoint-to-midpoint, midpoint-to-midpoint, or no intersection). If any pair of consecutively drawn segments differs (beyond thresholds) in two or more characteristics, the point between those segments is considered a *segment difference point*, a point of possible transition between a symbol and a connector. Candidate symbols are sequences of segments, bounded by two segment difference points, containing between two and the user-specific maximum number of segments.

Consider applying the approach to the voltage source in Figure 3. Although not shown, the segmentation is the obvious set of lines and an arc. There is a segment difference point between the lower, vertical line and the arc: the two segments differ in size and type. There is another segment difference point between the upper, vertical line and the horizontal line in the plus sign: they differ in size and intersection type (none vs. midpoint-to-midpoint). The segments between these two points constitute the voltage source.

Parsing Step 2: Pruning Based on Domain Knowledge: Not all of the symbols enumerated by our symbol locators are valid symbols. The final parsing step is to use domain specific information to prune out the candidates that are unlikely to be symbols. This is done using several heuristics. However, before the pruning begins, each candidate symbol must first be classified with our symbol recognizer, as the results of classification are used in the heuristics. The basic approach is to collect information supporting and refuting the fact that a group of segments is a symbol. The following is a summary of the heuristics we use for the electric circuit domain.

Indications that a group of segments may be an electrical component include:

- The ink density of the candidate component is high.
- The probability of match between the candidate component and the class identified by the recognizer is high.
- The candidate component contains enough pen strokes to be the component it was classified as. For example, a resistor can be drawn in a single stroke, but a current source requires at least two.
- The candidate component contains a full circle. Wires are never full circles.
- Two segments touching the candidate component are collinear. This is a good indication of a component because many components are drawn with collinear wires connected on each side.

Indications that a group of segments may not be an electrical component include:

- The bounding box of the candidate component is thin.
- The bounding box of the candidate component is large compared to that of other candidates.
- The average length of the segments in a candidate component is long. Components often contain many short segments, while wires are frequently long segments.
- The candidate component contains three or fewer segments that are all connected by endpoint to endpoint intersections. This pattern is typical of a wire that has been split into multiple segments. No standard components fit this description.
- The candidate component has the wrong number of connections for the component it has been classified as. For example, a ground symbol should have only one connection, a resistor should have two, and a transistor should have three.

Each candidate is assigned a heuristic score, which is initially zero. Points are added for positive indications, and are subtracted for negative indications. For a candidate to be considered a component, its heuristic score must be above a threshold. Additionally, because two symbols cannot share segments, any candidate overlapping another candidate with higher heuristic score is pruned. Any segments not identified as part of a symbol are considered to be connectors.

Symbol Recognition

The task of the symbol recognizer is to classify each candidate symbol. The recognizer takes as input the segments comprising a candidate symbol and returns the best definition. Our recognizer uses training examples to construct a probabilistic definition model of each symbol, based on geometric features of the segments. This probabilistic approach naturally accounts for the variations inherent in hand-drawn sketches and allows symbols to be drawn using any number of strokes drawn in any order. The recognizer is insensitive to size and orientation, and is robust to moderate non-uniform scaling.

Training: To train the recognizer, the user draws several examples of a symbol. The examples are segmented, and a set of nine geometric features are extracted from each example. These include the number of: pen strokes, line segments, arc segments, endpoint ("L") intersections, endpoint-to-midpoint ("T") intersections, midpoint ("X") intersections, pairs of parallel lines, and pairs of perpendicular lines. The final feature is the average distance between the endpoints of the segments, normalized by the maximum distance between any two endpoints. This feature helps differentiate between objects containing non-uniformly scaled versions of the same segments. For example, the average distance between the endpoints of a square is larger than that of a rectangle.

Once the values of the nine features have been determined for each of the training examples of a particular symbol, a statistical definition model is constructed. We assume a Gaussian distribution for the feature values, and thus the distribution is characterized by a mean and standard deviation. However, because eight of the features assume only discrete values, and since we aim to use only a handful of training examples, which may happen to have little difference in some features, continuous Gaussian models are not theoretically appropriate. Nevertheless, our empirical results show that these models produce highly favorable recognition rates.

Classification: We use a statistical classifier to determine which definition is the best match for an unknown symbol. The first step in recognizing an unknown symbol, S, is to extract the same nine features used to describe the training examples. The values of these features are then compared to the observed distributions of the features for each of the learned definitions, D_i. The unknown is classified by the definition that best matches it. Mathematically, the goal is to find the definition D^* that has the highest probability of matching S:

$$D^* = \arg\max_i P(D_i \mid S)$$

We assume that all definitions are equally likely to occur, and hence we set the prior probabilities of the definitions to be equal. We also assume that the nine geometric features x_j are independent of one another. Otherwise, a much larger number of training examples would be required for classification. Bayes' Rule tells us that the definition which best classifies the symbol is therefore the one that maximizes the likelihood of observing the symbol's individual features:

$$D^* = \arg\max_i \prod_j P(x_j \mid D_i)$$

As stated above, we assume each statistical definition model $P(x_j|D_i)$ to be a Gaussian distribution with mean $\mu_{i,j}$ and standard deviation $\sigma_{i,j}$.

$$P(x_j \mid D_i) = \frac{1}{\sigma_{i,j}\sqrt{2\pi}} \exp\left[\frac{-(x_j - \mu_{i,j})^2}{2\sigma_{i,j}^{\ 2}}\right]$$

Since we are assuming that the features are independent, this is referred to as a naïve Bayesian classifier. This type of classifier is commonly thought to produce optimal results only when all features are truly independent. This is not a proper assumption for our problem, since some of the features we use are interrelated. For example, the number of intersections in a symbol frequently increases with the number of lines and arcs. However, Domingos and Pazzani (1996) showed that the naïve Bayesian classifier does not require independence of the features to be optimal. While the actual probabilities of match may not be accurate, the rankings of the definitions will most likely be so.

Because of our assumption of a Gaussian distribution, definitions in which the training examples show no variation in one or more features cause difficulty during recognition. This situation is a common occurrence because a small number of training examples are often used, and because eight of the features used for classification can assume only discrete values. To prevent definitions from becoming overly rigid in this way, we require that all features, with the exception of the continuously valued average distance between endpoints, have a standard deviation of at least 0.3. This significantly increases recognition rates, especially when only a few training examples have been used.

Automated Error Correction

Once the parsing and recognition steps are complete, the system knows the locations of the symbols, and the connections between them. At this point, the system can use domain specific knowledge to correct parsing and recognition errors. Here we summarize our approach for circuits.

One indication that there may be a parsing problem in an electric circuit is that a large number of consecutively drawn segments have been identified as wires (Figure 5a). It is uncommon for a user to draw wires this way, thus suggesting that a component has been missed. In such situations, the system first tries to find the missed component by lowering the threshold for heuristic pruning. If a component is still not found, a miss-classification may have caused the parser to err. In this case, the system considers lower ranked classifications for any candidate components that contain the wire segments in question. If the score of one of those candidates is now above the heuristic threshold, the system keeps that candidate and its new classification.

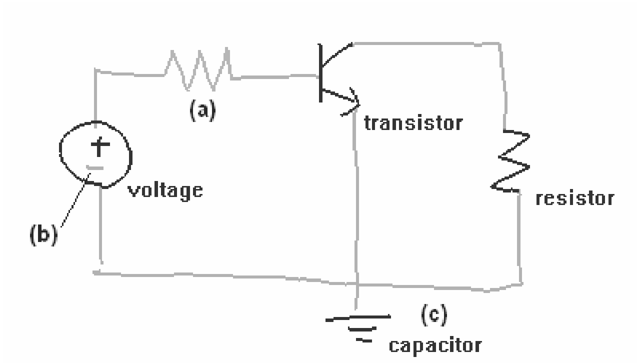

Figure 5: Errors the system can find and repair. (a) Missed component diagnosed as a wire with too many consecutive segments. (b) Parsing error diagnosed as a dangling wire. Here, segment has been identified as a wire, but it has no connections. (c) Recognition error diagnosed by considering connection count. Capacitors require two connections, but this component has only one.

Because wires are used to connect components, it is unlikely that a wire will have fewer than two connections (Figure 5b). If a wire is found to have only one connection, the system first checks to see if it should belong to a nearby component. If not, the ends of the wire are extended to see if any more connections are found.

Another indication of a problem is that a component has the wrong number of connections (Figure 5c). This is often a result of an incorrect classification by the recognizer. The problem is sometimes fixed by selecting the second choice of the recognizer. Otherwise, we assume that the problem is due to the sketchiness of the drawing – two segments that were intended to intersect did not, or two segments that were not supposed to intersect did. To fix this, the component's segments, and the nearby wire segments, are extended or shortened until the correct number of connections is found.

Results and Discussion

We conducted a formal user evaluation of AC-SPARC. Ten users were each asked to draw eight specific circuits containing on average 9.25 electrical components. Users pre-trained the recognizer by providing six examples of each of the components used in the circuits. This included resistors, inductors, capacitors, transistors, voltage sources, current sources, and grounds. Users drew in the raw ink mode, and could not see the segmentation.

The system performed well for all but one user, who drew in a particularly sloppy fashion. Excluding this user, the program correctly located and recognized 81% of the symbols in the sketches. The average for all ten users was 77%. On average, it took users only 2.6 editing gestures per sketch to correct errors (see "System Overview"). Note that these results are for users with no prior experience with our system. We have found that as users become more familiar with our system, even better performance is achieved.

Overall, 91% of the symbols were segmented correctly. Of the correctly segmented symbols, 86.7% were parsed correctly, and 95.4% of the correctly parsed symbols were recognized correctly. Thus, while all phases of processing performed reasonably well, symbol recognition was responsible for the fewest errors, and parsing was responsible for the most.

AC-SPARC uses a default value of one for each of the parameters of the circuit components. For example, resistors are assigned a default resistance of 1 ohm. We plan to add functionality enabling the user to specify the parameter values with the stylus. One approach would be to provide a property window (dialog box) that is accessed by tapping the stylus on a symbol. When the window appears, the user would write the desired parameter value, which would then be recognized with a handwriting recognizer.

AC-SPARC was intended as a test bed for our parsing and recognition techniques. To make a more useful engineering tool, it is necessary to improve some of the basic user interface features. For example, the system offers a cut (erase) function, but needs copy and paste functions. Likewise, the editing gestures were designed for ease of programming, and need to be made more flexible and robust.

Related Work

Graph-based methods have recently been applied to the problem of symbol recognition. Symbols are segmented into geometric primitives, and graphs encode both the intrinsic attributes of the primitives and the geometric relationships between them. Recognition is formulated as a graph-subgraph isomorphism problem. In Lee's (1992) approach, the graph represents precise geometry, and thus the approach is suitable for precisely drawn symbols. Calhoun *et al.* (2002) developed an approach in which the graph encodes topology, rather than geometry, so as to be more tolerant of drawing variations. Graph-based approaches are typically sensitive to segmentation errors, and graph matching can be expensive. Our recognizer avoids the cost of graph matching by encoding topology as features, and our probabilistic approach is robust to drawing variations and segmentation errors.

There are a variety of other feature-based recognition approaches. Typically, the features describe aggregate properties of the symbol, rather than the topology. However, different shapes can have the same aggregate features, resulting in miss-classification. Rubine's (1991) trainable gesture recognizer is suitable for single stroke gestures (symbols) drawn in preferred orientations. Symbols are described by eleven geometric and two dynamic features. Apte *et al.* (1993) developed a hand-coded recognizer based on geometric properties of the convex hull of a symbol. Likewise, Fonseca *et al.* (2002) developed a method that uses a naïve Bayesian classifier to recognize multi-stroke shapes described by their convex hulls.

Kara and Stahovich (2004a) present a multi-stroke recognizer based on a down-sampled bitmap representation. The approach is well suited to "sketchy" drawings, such as those with over-tracing. However, as the method is based on geometry not topology, it is sensitive to non-uniform scaling and large variations in shape.

The problem of sketch parsing is beginning to draw attention from researchers. For example, Saund *et al.* (2002) present a system that uses Gestalt principles to locate salient objects in a sketch. The system is intended to assist users in interactively manipulating objects on a drawing surface. Shilman *et al.* (2002) present a statistical visual language model for ink parsing. The approach requires a manually coded visual grammar and assumes that the lowest level objects can be recognized in isolation. Alvarado (2003) proposed a parsing approach based on dynamically constructed Bayesian networks. The approach is similar to Shilman's, but the lowest level objects are geometric primitives, rather than symbols that must be recognizable in isolation. Kara and Stahovich (2004b) present a parsing approach based on a "mark-group-recognize" architecture. First, a preliminary recognizer is used to identify "marker symbols," symbols that have unique geometric and kinematic properties that allow them to be easily extracted from a continuous stream of input. These are then used to efficiently cluster the remaining strokes into symbols.

There have been recent efforts to create general purpose sketch understanding tools. For example, Hong and Landay (2000) created a generalized sketching and gesturing toolkit called SATIN that eliminates the needless re-implementation of the basic functionalities typical of pen-based applications. Similarly, Mankoff *et al.* (2000) have created a set of general purpose ambiguity resolution strategies, called mediation techniques. They demonstrated these techniques in a program called Burlap.

In recent years, experimental sketch-based interfaces have been developed for a number of different disciplines.

Landay and Myers (2001) present an interactive sketching tool called SILK that allows designers to quickly sketch a user interface and transform it into a functional system. As the designer sketches, SILK's recognizer identifies the user interface component represented by each pen stroke. Alvarado (2000) presents a system called ASSIST that can interpret and simulate a variety of hand-drawn mechanical devices. A key strength of this system is its ability to augment implicit and explicit user feedback with contextual information to disambiguate between multiple interpretations of a drawing.

A few sketch-based interfaces have been developed for electrical circuits. Narayanaswamy's (1996) SPICE interface uses a hard-coded recognizer that assumes a fixed drawing order. The system requires the user to pause between symbols. Hong and Landay (2000) demonstrated the capabilities of their SATIN system by creating Sketchy-SPICE, a circuit CAD tool for circuits containing AND, OR, and NOT gates. The gates must be drawn in either one or two strokes. Lee (1992) describes a trainable recognizer for electrical circuit symbols. This recognizer requires that each symbol be drawn using only one or two strokes.

Conclusions

We have presented a sketch parsing technique that can automatically extract hand-drawn symbols from a continuous stream of pen strokes. The user is not required to provide explicit indications of where symbols start and end, but must complete drawing one symbol before starting the next. Our technique is suitable for network-like diagrams containing isolated, non-overlapping symbols that are linked together. We have also developed a probabilistic, feature-based symbol recognizer. Our recognition technique provides several advantages: it is insensitive to rotations and scaling, the parts of a symbol can be drawn in any order, symbols can be drawn in multiple pen strokes, and a single pen stroke can contain multiple symbols.

Due to the variations, inconsistencies, and ambiguities inherent in hand-drawn sketches, it is difficult to achieve perfect parsing and recognition accuracy. Our sketch interpreter employs automatic error correction techniques to help fix typical errors. Once a sketch has been interpreted, domain knowledge is used to determine if the interpretation of the sketch is self-consistent. If not, parsing and recognition are revisited so as to eliminate the inconsistencies.

We have used these techniques to build AC-SPARC, a sketch-based user interface for the SPICE circuit analysis program. We conducted a user study to evaluate the performance of AC-SPARC, and the results were promising. While AC-SPARC can be used to solve real problems, it clearly requires refinement before it can serve as a production engineering tool. Nevertheless, our system has demonstrated that it is possible to create an interface that combines the ease of pencil and paper sketching with the power of traditional computer software.

References

Alvarado, C. (2000). A Natural Sketching Environment: Bringing the Computer into Early Stages of Mechanical Design. Masters Thesis, MIT.

Alvarado, C. (2003). Dynamically constructed Bayesian networks for sketch understanding. Technical Report, MIT Project Oxygen Student Workshop.

Apte, A., Vo, V., & Kimura, T.D. (1993). Proc. *16th ACM Symposium on User Interface Software & Technology*, pp. 121-128.

Calhoun, C., Stahovich, T.F., Kurtoglu, T., & Kara, L.B. (2002). Recognizing multi-stroke symbols. Proc. *AAAI 2002 Spring Symp. on Sketch Understanding*, pp. 15-23.

Domingos, P., & Pazzani, M. (1996). Beyond Independence: Conditions for the Optimality of the Simple Bayesian Classifier. Proc. *Thirteenth International Conference on Machine Learning*, pp. 105-112.

Fonseca, M., Pimentel, C., & Jorge, J. (2002). CALI: An Online Scribble Recognizer for Calligraphic Interfaces. Proc. *AAAI 2002 Spring Symposium on Sketch Understanding*, pp. 51-58.

Hong, J.I., & Landay, J.A. (2000). Satin: A toolkit for informal ink-based applications. Proc. *ACM Conference on User Interfaces and Software Technology*, pp. 63-72.

Kara, L. B. and Stahovich, T. F. (2004a). An Image-Based Trainable Symbol Recognizer for Sketch-Based Interfaces. Proc. *AAAI 2004 Fall Symposium on Making Pen-Based Interaction Intelligent and Natural.*

Kara, L. B. and Stahovich, T. F. (2004b). Hierarchical Parsing and Recognition of Hand-Sketched Diagrams. Proc. *Seventeenth Annual ACM Symposium on User Interface Software and Technology, (UIST 2004).*

Landay, J.A, & Myers, B.A. (2001). Sketching interfaces: Toward more human interface design. *IEEE Computer*, 34(3), 56-64.

Lee, S. (1992). Recognizing hand-drawn electrical circuit symbols with attributed graph matching. In *Structured Document Image Analysis* (Baird, H.S., Bunke, H., & Yamamoto, K., Eds.), pp. 340-358. Springer-Verlag.

Mankoff, J., Abowd, G., & Hudson, S. (2000). OOPS: A toolkit supporting mediation techniques for resolving ambiguity in recognition-based interfaces. *Computers and Graphics*, 24(6), 819-834.

Narayanaswamy, S. (1996). Pen and Speech Recognition in the User Interface for Mobile Multimedia Terminals. Ph.D. Thesis, University of California at Berkeley.

Rubine, D. (1991). Specifying gestures by example. *Computer Graphics*, 25(4), 329-337.

Saund, E., Mahoney, J., Fleet, D., Larner, D., & Lank, E. (2002). Perceptual organization as a foundation for intelligent sketch editing. Proc. *AAAI 2002 Spring Symposium on Sketch Understanding*, pp. 118-125.

Shilman, M., Pasula, H., Russell, S., & Newton, R. (2002). Proc. *AAAI 2002 Spring Symposium on Sketch Understanding*, pp. 126-132.

Stahovich, T. F. (2004). Segmentation of pen strokes using pen speed. Proc. *AAAI 2004 Fall Symposium on Making Pen-Based Interaction Intelligent and Natural.*